NEWS & BUZZ

In-Place Pod Resize in Kubernetes: Enabling Non-Disruptive Vertical Scaling

Joaquín Entrialgo[1,*]

[1] Department of Informatic Systems, Polytechnical University of Madrid, Madrid 28030, Spain

Abstract

Kubernetes now supports In-Place Pod Resize, a mechanism that allows CPU and memory adjustments without restarting Pods. This removes a key limitation of vertical scaling and reduces disruption for stateful or latency-sensitive workloads. In this News & Buzz article, we explain how the feature works, its integration with the Kubelet and container runtimes, and its impact on the Vertical Pod Autoscaler and hybrid autoscaling. Finally, we highlight open research directions, including scheduler coordination, application adaptation, and real-world performance evaluation.

Keywords: vertical autoscaling, kubernetes, containers, cloud-native.

1 Introduction

Kubernetes has become de facto standard for container orchestration in cloud-native applications and distributed microservice systems, and it has recently announced [1] a significant addition to its autoscaling technologies: in-place pod resize. Kubernetes provides inherent support for scaling containerized workloads, primarily through the Horizontal Pod Autoscaler (HPA), which adjusts the number of Pod replicas (horizontal scaling), and the Vertical Pod Autoscaler (VPA), which computes and recommends resource allocations for the constituent containers within an existing Pod (vertical scaling).

A significant architectural barrier for effective vertical scaling, particularly when implemented via the VPA in its traditional modes (e.g., Recreate mode), was the need to restart the Pod whenever its resource settings had to be updated [2]. Because the Pod specification is immutable, adjusting a container's resource allocation required terminating its encompassing Pod and creating a new one. This mandatory recreation presents substantial operational drawbacks for disruption-sensitive workloads, including stateful applications (such as databases or message queues) and long-running batch processes, risking service disruption, loss of in-progress computation, and complexity in production environments governed by stringent Service Level Objectives (SLOs) [3].

The introduction of In-Place Pod Resize, also called In-Place Pod Vertical Scaling (IPVS), represents a significant improvement in the resource management capabilities of Kubernetes, offering a mechanism to dynamically modify container CPU and memory resources without requiring a Pod restart. This feature fundamentally enhances system scalability by enabling

Submitted: 21 November 2025
Accepted: 25 March 2026
Published: 30 March 2026

Vol. 1, **No.** 1, 2026.
10.62762/JSS.2025.776624

***Corresponding author:**
✉ Joaquín Entrialgo
j.entrialgo@upm.es

Citation
Entrialgo, J. (2026). In-Place Pod Resize in Kubernetes: Enabling Non-Disruptive Vertical Scaling. *Journal of Systems Scalability*, 1(1), 35–38.

© 2026 by the Author. Published by Institute of Central Computation and Knowledge. This is an open access article under the CC BY license (https://creativecommons.org/licenses/by/4.0/).

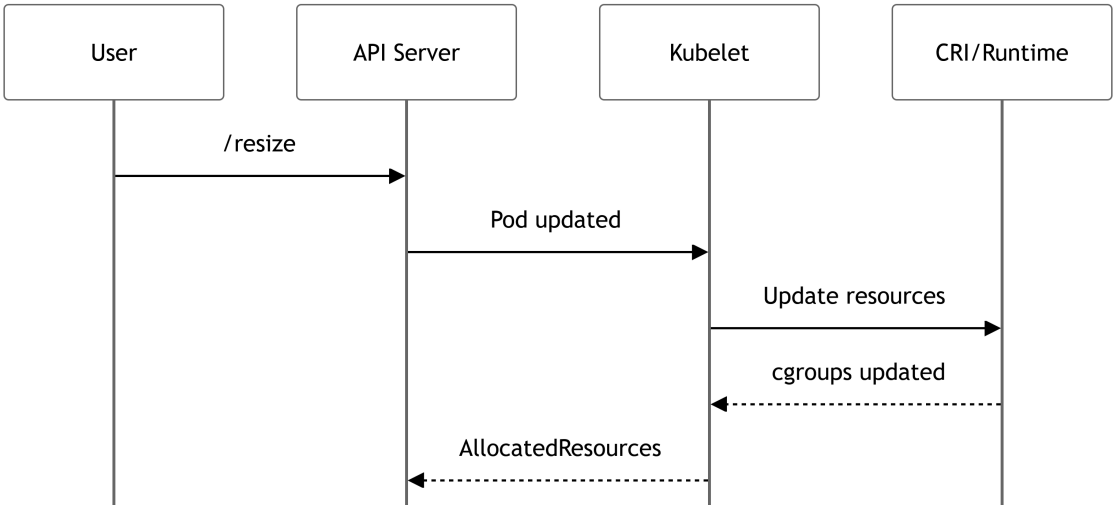

Figure 1. In-Place Pod Resize workflow.

fine-grained, non-disruptive resource adjustments, thereby improving resource utilization efficiency and responsiveness to transient workload variations.

2 The Technical Mechanism of In-Place Pod Resize

First introduced as an alpha feature in Kubernetes v1.27, In-Place Pod Resize progressed to Beta status and was enabled by default in Kubernetes v1.33 and finally graduated to stable in Kubernetes v1.35. This functionality addresses the long-standing limitations imposed by the immutability of the Pod specification's resources field. The feature is tracked by the Kubernetes community under Kubernetes Enhancement Proposal (KEP) 1287 [4]. The core mechanism operates through several architectural changes:

1. **Mutability and API Interaction:** The `spec.containers[*].resources` field, specifically for CPU and memory, has been made mutable. Resource updates are performed by interacting with a new dedicated `/resize` subresource in the Kubernetes API.

2. **Kubelet and CRI Handshake:** The Kubelet (the agent responsible for managing nodes) intercepts the resize request and communicates the resource adjustment to the container runtime (e.g., containerd or CRI-O) via the Container Runtime Interface (CRI). The runtime subsequently adjusts the container's control groups (cgroups) asynchronously. Figure 1 summarizes this workflow.

3. **State Tracking and Status Conditions:** The feature introduces sophisticated status tracking to reflect the state of the resource allocation. The field `Pod.Status.ContainerStatuses[i].AllocatedResources` reflects the resources currently allocated to the container, serving as the source of truth for the container runtime. Furthermore, new Pod conditions indicate the resize status:

   - **PodResizePending:** Indicates that the Kubelet cannot immediately fulfill the request, often due to insufficient node resources or temporary deferral.
   - **PodResizeInProgress:** Indicates the resize request has been accepted and is actively being applied.

4. **Resize Policy:** To afford granular control over container behavior during resizing, the `resizePolicy` field can be defined per resource. While `NotRequired` (the default for CPU) attempts in-place scaling, `RestartContainer` forces a container restart for the resource change (often necessary for memory adjustments or when constrained by cgroup versions).

3 Implications for Autoscaling Research and Practice

The introduction of In-Place Pod Resize changes the design space for autoscaling research:

- **Vertical Pod Autoscaler (VPA) Enhancement:** The VPA traditionally faced limitations due to its reliance on pod eviction. To integrate the core In-Place Pod Resize capability into the VPA, a new update mode, `InPlaceOrRecreate`, was added.

This mode prioritizes non-disruptive, in-place resizing and only resorts to recreation/eviction if the resource constraints of the node render the in-place update infeasible (e.g., when the required resource increase exceeds the remaining capacity of the current node). This promises to unlock wider adoption of VPA for stateful and high-availability workloads.

- **Hybrid and Multi-Metric Autoscaling:** The non-disruptive nature of vertical scaling is essential for sophisticated autoscaling frameworks that combine horizontal and vertical strategies. Systems like KOSMOS [5], designed for multi-level control and in-place vertical scaling, leverage this capability to quickly reconfigure container resources at runtime, using control-theoretical planners and heuristics to manage resource contention. In-Place Pod Resize directly supports the implementation of hybrid autoscalers, enabling them to flexibly trade off latency (often optimized by horizontal scaling) and CPU utilization/cost (optimized by vertical scaling) [6].
- **Resource Efficiency and Cost Optimization:** By enabling real-time resource tuning, In-Place Pod Resize minimizes the need for defensive over-provisioning, a common cause of cloud waste and increased costs [7].

4 Future Research Directions

While the graduation of In-Place Pod Resize to Stable status signifies robust foundational stability, several areas require further research to leverage its full potential in scalable systems:

1. **Multi-Dimensional Constraint Optimization:** Current In-Place Pod Resize support is constrained primarily to CPU and memory. Future research must investigate mechanisms to extend non-disruptive scaling to other critical resources, such as GPUs, ephemeral storage, and network I/O, particularly for computation-intensive tasks like Machine Learning pipelines or complex data streaming applications.
2. **Scheduler Integration:** Optimizing the scheduling layer to support in-place Pod resize requires making the scheduler explicitly aware of Pods in the `ResizePending` state and their updated resource demands. When a resize request cannot be immediately satisfied on the current node, the Pod remains running with its original allocation but is logically treated as requiring the desired resources for future placement decisions. In this situation, the scheduler evaluates feasibility using, for each resource dimension, the maximum of: (i) the Pod's desired resources (post-resize), (ii) the currently allocated resources, and (iii) the actual runtime usage. This conservative estimate prevents underestimation of resource requirements and avoids overcommitment during rescheduling. However, the current design does not provide explicit reservation of resources ("headroom") for pending resizes. As a result, resize requests may be delayed or may eventually require Pod eviction and rescheduling if sufficient capacity is not available. Future work should therefore explore mechanisms to proactively reserve or guarantee headroom for Pods with pending resizes. Possible approaches include scheduler-level resource reservation, priority-aware preemption policies, or integration with cluster autoscaling to ensure that sufficient capacity is available when the resize is applied. Such mechanisms would reduce the likelihood of resize-induced disruptions and improve overall system stability.
3. **Advanced Hybrid Policy Development:** The foundation laid by In-Place Pod Resize facilitates the design of advanced proactive and predictive autoscaling policies that utilize machine learning and control theory. Research should prioritize integrating In-Place Pod Resize within SLO-aware scaling systems (e.g., setting scaling targets based on application latency or queue depth, rather than basic utilization metrics), and exploring ways to safely relax limitations, such as constraints on decreasing memory limits.
4. **Performance and Overheads Quantification:** While community design documents anticipate improvement efficiency and easier autoscaling, rigorous, large-scale evaluations remain an open challenge. Empirical evaluations are required to precisely quantify the latency, overhead, and potential performance impact of in-place resize operations compared to traditional recreation across diverse container runtimes and heterogeneous workloads. This is essential for informing the continuous co-design process between applications and evolving cloud platforms.

5. **Addressing the Application Awareness Gap:** Many applications, particularly those ported from a pre-container world, are not "cgroup-aware." They do not dynamically poll the kernel for their current resource limits. Instead, they inspect their environment once, at startup, and configure their internal memory pools based on those initial values. For instance, Java Virtual Machines (JVMs) allocate a fixed heap size at startup and will crash if the underlying cgroup memory is aggressively scaled down beneath that heap size. This can create two problems: 1) in scaling up memory, the application remains unaware of the new available memory and, therefore, it doesn't use it; and 2) in scaling down memory, the application continues to operate, believing that it has more memory than it really has: if it tries to access outside of the new limits, the kernel will identify it as a rogue process and terminate it instantly.

5 Conclusion

In-Place Pod Resize strengthens Kubernetes' ability to adjust resources without interrupting running workloads. Its arrival makes vertical scaling more practical in real deployments and creates new opportunities to combine it with existing autoscaling techniques. At the same time, it raises open questions about scheduler behavior, application awareness, and real-world performance that deserve further study.

Data Availability Statement

Not applicable.

Funding

This work was funded by the the Spanish National Plan for Research, Development and Innovation from the Spanish Ministerio de Ciencia e Innovación under Grant PID2024-155752OB-I00.

Conflicts of Interest

The author declares no conflicts of interest.

AI Use Statement

The author declares that generative artificial intelligence (AI) was used solely for language editing to improve the clarity and readability of the manuscript. The AI tool utilized was ChatGPT-5. The authors take full responsibility for the content of the manuscript, including the accuracy of the information and the integrity of the research.

Ethical Approval and Consent to Participate

Not applicable.

References

[1] Sarkar, N. (2025, May 16). Kubernetes 1.35: In-Place Pod Resize graduates to stable. Kubernetes Blog. Retrieved from https://kubernetes.io/blog/2025/12/19/kubernetes-v1-35-in-place-pod-resize-ga/

[2] Kubernetes. (n.d.). In-place updates support (Enhancement #4016). GitHub. Retrieved November 20, 2025, from https://github.com/kubernetes/autoscaler/tree/master/vertical-pod-autoscaler/enhancements/4016-in-place-updates-support

[3] Na, J. H., Yu, H. J., Kang, H., Kang, H., Lim, H. D., Shin, J. H., & Noh, S. Y. (2024). PVA: the persistent volume autoscaler for stateful applications in kubernetes. *IEEE Access, 12*, 179130-179143. [CrossRef]

[4] Kubernetes. (2025). In-place pod resize (KEP-1287). GitHub. Retrieved November 20, 2025, from https://github.com/kubernetes/enhancements/issues/1287

[5] Baresi, L., Hu, D. Y. X., Quattrocchi, G., & Terracciano, L. (2021, November). KOSMOS: Vertical and horizontal resource autoscaling for kubernetes. In *International Conference on Service-Oriented Computing* (pp. 821-829). Cham: Springer International Publishing. [CrossRef]

[6] Feng, Y., Li, J., Li, L., Li, H., Xiao, Z., & Wang, X. (2025, August). HyMetricScaler: A Multi-Metric-Driven Hybrid Autoscaling Framework for Kubernetes. In *2025 IEEE International Conference on High Performance Computing and Communications (HPCC)* (pp. 146-153). IEEE. [CrossRef]

[7] Marchese, A., & Tomarchio, O. (2025, July). SLO-aware container orchestration on Kubernetes clusters. In *2025 IEEE 18th International Conference on Cloud Computing (CLOUD)* (pp. 318-327). IEEE. [CrossRef]

Joaquín Entrialgo received the Ph.D. degree in Computer Science from the University of Oviedo, Spain, in 2006. He is currently an Associate Professor at the Technical University of Madrid and was previously with the University of Oviedo. His early research focused on timing analysis for real-time systems, including probabilistic approaches. He later worked on performance and quality-of-service evaluation in web servers, followed by research on energy-efficient computing in both servers and client devices. His current research centers on the optimization of cloud and edge infrastructures. (Email: j.entrialgo@upm.es)